allinea

SCALE TO NEW HEIGHTS

Debugging the Future with DDT at ORNL

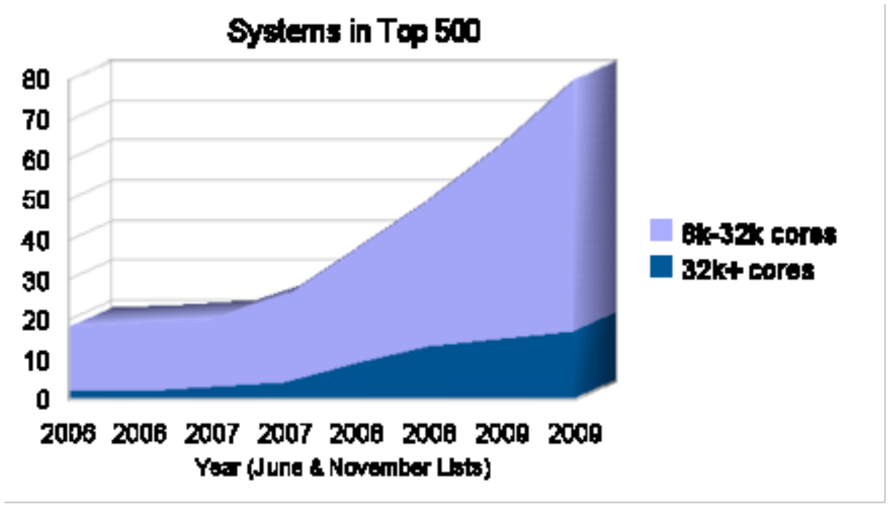
David Lecomber

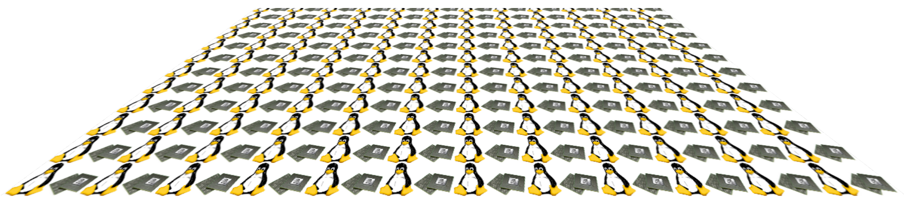
david@allinea.com

CTO

Interesting Times ...

- Processor counts growing rapidly
- GPUs entering HPC
- Large hybrid systems imminent
- But what happens when software doesn't work?





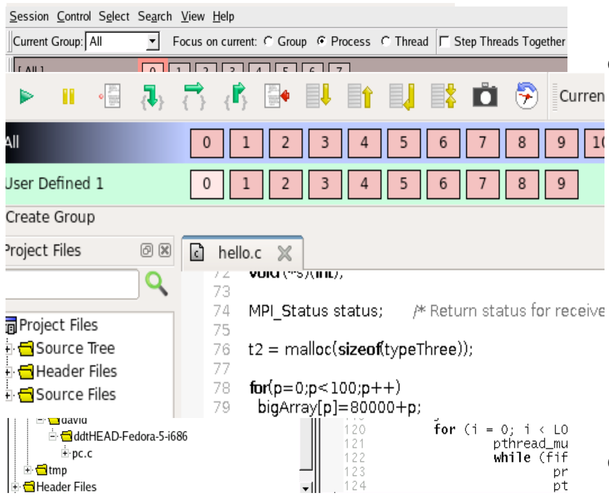
About Allinea

- A company focussed on HPC tools
  - DDT – the easiest tool for debugging parallel codes at every scale
  - OPT – instruments code to find bottlenecks
  - DDTLite – plug-in for Visual Studio 2008

- Significance of previous graph?
  - **Everyone** aspires to have code running on more cores
  - **No-one** can debug whole-machine jobs on any of these systems
  - **How many systems next year ...?**

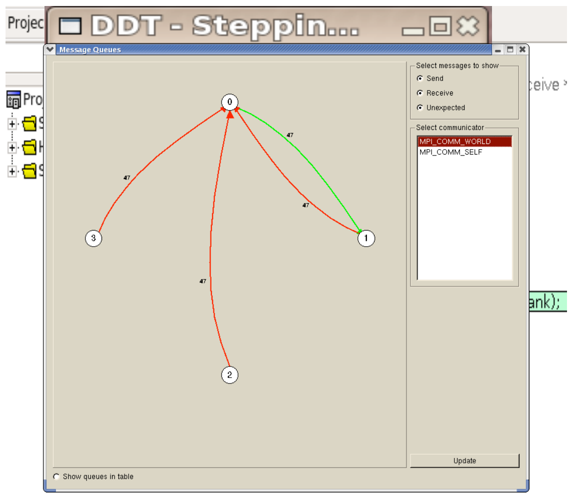
allinea

DDT for everyone



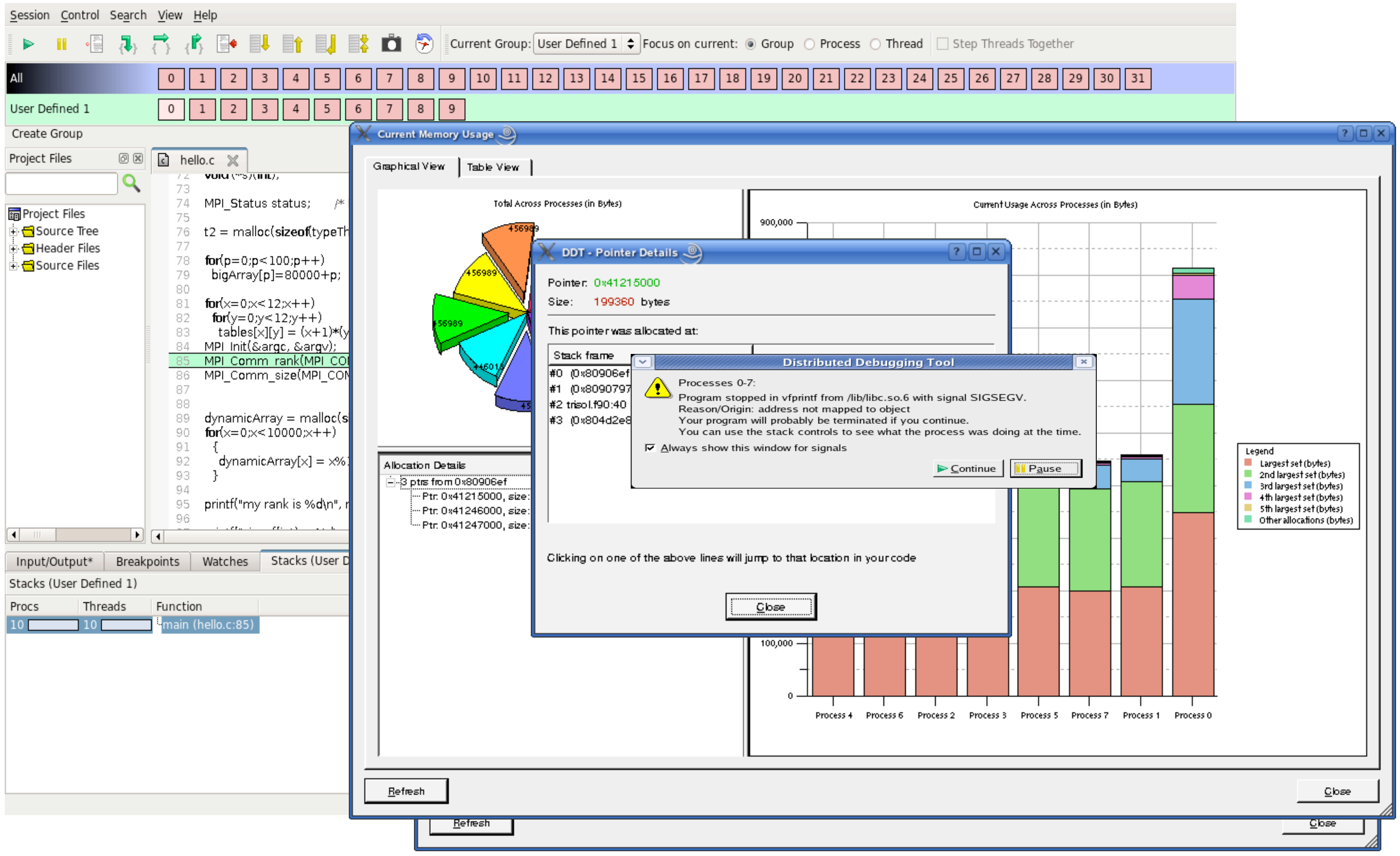
```
void (*s)(int);

MPI_Status status;    /* Return status for receive

t2 = malloc(sizeof(typeThree));

for(p=0;p<100;p++)
 bigArray[p]=80000+p;
```

```
            for (i = 0; i < LO
                    pthread_mu
                    while (fif
                            pr
                            pt
```



- Scalar features
  - Advanced C++ and STL
  - Fortran 90, 95 and 2003: modules, allocatable data, pointers, derived types
  - Memory debugging
- Multithreading & OpenMP features
  - Step, breakpoint etc. one or all threads
- MPI features
  - Easy to manage groups
  - Control processes by groups
  - Compare data
  - Visualize message queues (not Cray!)

Memory Debugging



```
void (*s)(int);

MPI_Status status;      /*

t2 = malloc(sizeof(typeTh

for(p=0;p<100;p++)
  bigArray[p]=80000+p;

for(x=0;x<12;x++)
  for(y=0;y<12;y++)
    tables[x][y] = (x+1)*(y
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_CO
MPI_Comm_size(MPI_CON


dynamicArray = malloc(s
for(x=0;x<10000;x++)
  {
    dynamicArray[x] = x%1
  }

printf("my rank is %d\n", r
```

Stacks (User Defined 1)

Procs	Threads	Function
10	10	main (hello.c:85)

Allocation Details
- 3 ptrs from 0x80906ef
  - Ptr: 0x41215000, size:
  - Ptr: 0x41246000, size:
  - Ptr: 0x41247000, size:

DDT - Pointer Details

Pointer: 0x41215000
Size: 199360 bytes

This pointer was allocated at:

Stack frame
#0 (0x80906ef
#1 (0x8090797
#2 trisol.f90:40
#3 (0x804d2e8

Clicking on one of the above lines will jump to that location in your code

Distributed Debugging Tool

Processes 0-7:
Program stopped in vfprintf from /lib/libc.so.6 with signal SIGSEGV.
Reason/Origin: address not mapped to object
Your program will probably be terminated if you continue.
You can use the stack controls to see what the process was doing at the time.

[✓] Always show this window for signals

Continue Pause

Full-strength Debugging

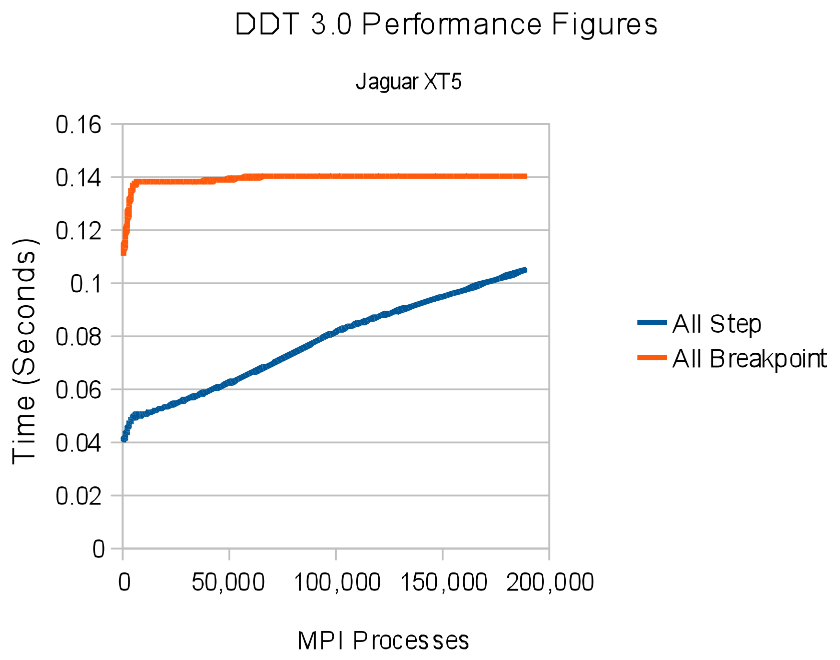
- Many benefits to graphical parallel debuggers
  - Large feature sets for common bugs
  - Richness of user interface and real control of processes
- Historically **all** parallel debuggers hit scale problems
  - Bottleneck at the frontend: Direct GUI → nodes architectures
    - Linear performance in number of processes
  - Human factors limit – mouse fatigue and brain overload
- Are tools ready for the task?
  - Allinea is changing the game!

Petascale Tools Project

- Allinea is developing a petascale debugging tool
  - Production Grade debugger
  - Multi-year project – with usable intermediate results
  - Commenced June 2009 – showing results already
- Building a multi-level tree for debugging
  - One tool from 1 to 250,000 cores
  - Goal of logarithmic performance scaling
- Scaling all aspects of debugging
  - Step, attach, data checking, ...
  - Many challenges ahead!

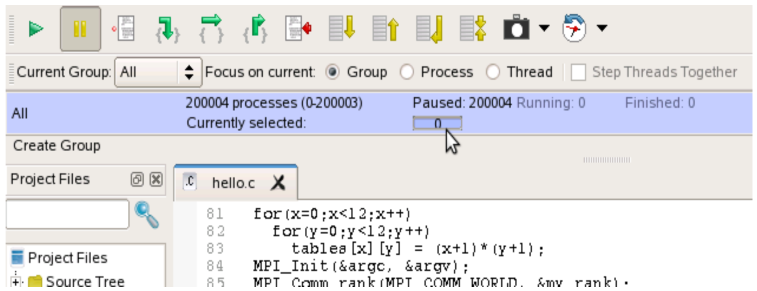
allinea

DDT: Petascale Debugging



- DDT is delivering petascale debugging **today**
  - A collaboration with ORNL on Jaguar Cray XT
  - Tree architecture – logarithmic performance
  - Many operations now faster at 220,000 than previously at 1,000 cores
  - **~1/10th of a second** to step and gather all stacks at 220,000 cores

Scalable Process Control



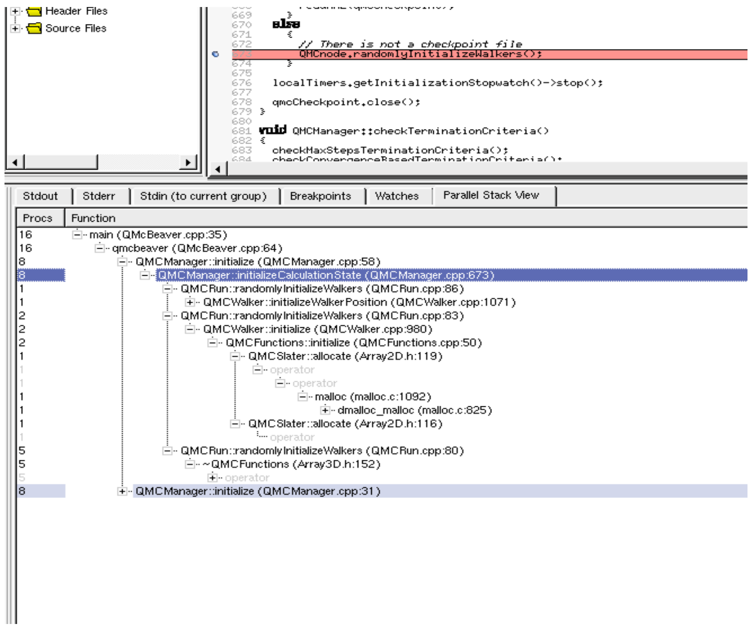
Current Group: All | Focus on current: [✓ Group] Process Thread | Step Threads Together

All — 200004 processes (0-200003) — Paused: 200004 Running: 0 Finished: 0 — Currently selected: 0

Create Group

Project Files — hello.c

```
for(x=0;x<12;x++)
  for(y=0;y<12;y++)
    tables[x][y] = (x+1)*(y+1);
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
```

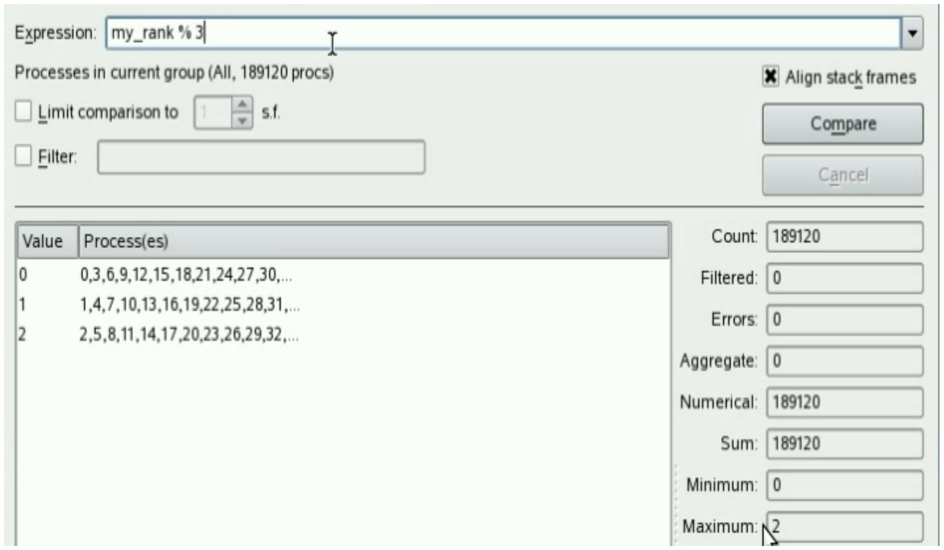


```
    }
  else
    {
      // There is not a checkpoint file
      QMCnode.randomlyInitializeWalkers();
    }

  localTimers.getInitializationStopwatch()->stop();

  qmcCheckpoint.close();
}

void QMCManager::checkTerminationCriteria()
{
  checkMaxStepsTerminationCriteria();
  checkConvergenceBasedTerminationCriteria();
```

```
Procs   Function
16      main (QMcBeaver.cpp:35)
16        qmcbeaver (QMcBeaver.cpp:64)
8           QMCManager::initialize (QMCManager.cpp:58)
8             QMCManager::initializeCalculationState (QMCManager.cpp:673)
1               QMCRun::randomlyInitializeWalkers (QMCRun.cpp:86)
1                 QMCWalker::initializeWalkerPosition (QMCWalker.cpp:1071)
2               QMCRun::randomlyInitializeWalkers (QMCRun.cpp:83)
2                 QMCWalker::initialize (QMCWalker.cpp:980)
2                   QMCFunctions::initialize (QMCFunctions.cpp:50)
1                     QMCSlater::allocate (Array2D.h:119)
1                       operator
1                         operator
1                           malloc (malloc.c:1092)
1                             dmalloc_malloc (malloc.c:825)
1                     QMCSlater::allocate (Array2D.h:116)
1                       operator
5               QMCRun::randomlyInitializeWalkers (QMCRun.cpp:80)
5                 ~QMCFunctions (Array3D.h:152)
5                   operator
8           QMCManager::initialize (QMCManager.cpp:31)
```

- Control Processes by Groups
  - Set breakpoints, step, play, stop etc. using user-defined groups
  - Scalable process groups view
  - Compact representation
- Parallel Stack View
  - Finds rogue processes faster
  - Identifies classes of process behaviour
  - Allows rapid grouping of processes

Presenting Data, Usefully


Expression: my_rank % 3

Processes in current group (All, 189120 procs)

☐ Limit comparison to 1 s.f.

☐ Filter:

☒ Align stack frames

Value	Process(es)
0	0,3,6,9,12,15,18,21,24,27,30,...
1	1,4,7,10,13,16,19,22,25,28,31,...
2	2,5,8,11,14,17,20,23,26,29,32,...

Count:	189120
Filtered:	0
Errors:	0
Aggregate:	0
Numerical:	189120
Sum:	189120
Minimum:	0
Maximum:	2

- Gather from every node
  - Potentially costly – if all data different
  - ... easy if values mostly same
  - New ideas
    - Aggregated statistics
    - Probabilistic algorithms optimize performance – even in pathological case
  - With a fast and scalable infrastructure, new things become possible
    - Watch this space!

Where Next?

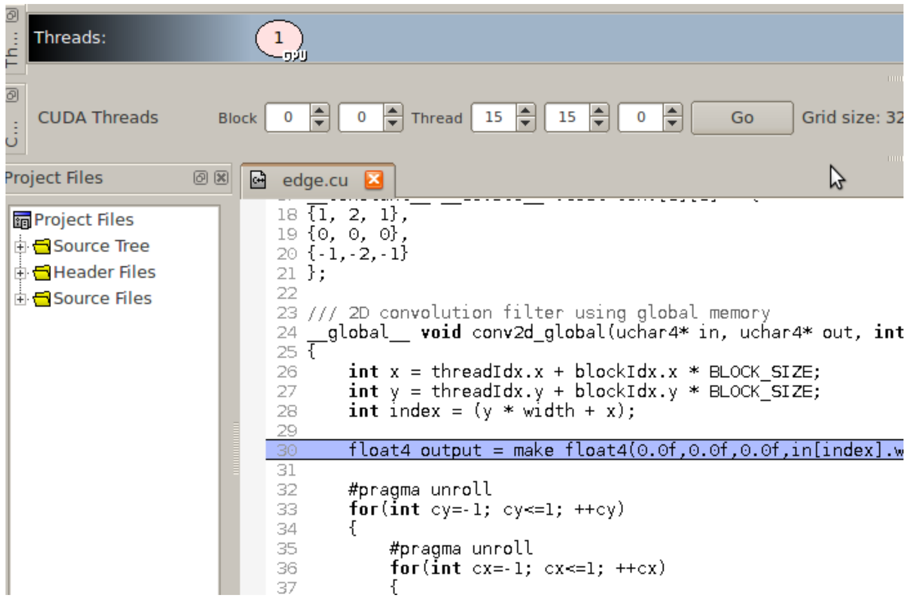
- DDT is the first Petascale debugger..
  - A debugging tool has finally caught up with the hardware!
    - Work is in progress to port every feature for scale
    - Memory debugging, data visualization, ...
  - How can the infrastructure be built upon?
    - Does DDT offer the right framework for collaboration?
    - Can we encourage a codebase of user-generated MPI tools/utilities?

- ... but large clusters are a fraction of HPC
  - Most parallel development starts smaller
  - Is now starting even smaller: GPUs

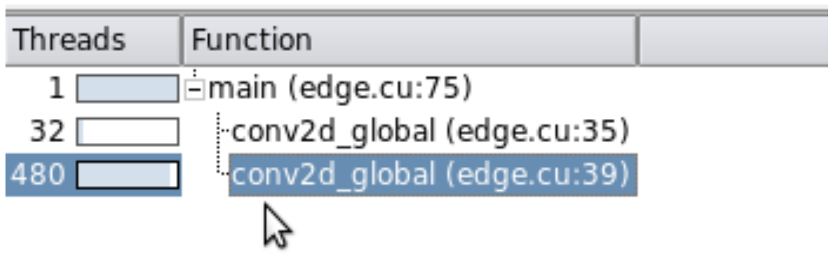
allinea

CUDA Threads in DDT

- Run the code
  - Browse source
  - Set breakpoints
  - Stop at a line of CUDA code
  - Stops once for each scheduled collection of blocks
- Select a CUDA thread
  - Examine variables and shared memory
  - Step a warp
  - View all extant threads in parallel tree view



```
{1, 2, 1},
{0, 0, 0},
{-1,-2,-1}
};

/// 2D convolution filter using global memory
__global__ void conv2d_global(uchar4* in, uchar4* out, int
{
    int x = threadIdx.x + blockIdx.x * BLOCK_SIZE;
    int y = threadIdx.y + blockIdx.y * BLOCK_SIZE;
    int index = (y * width + x);

    float4 output = make_float4(0.0f,0.0f,0.0f,in[index].w

    #pragma unroll
    for(int cy=-1; cy<=1; ++cy)
    {
        #pragma unroll
        for(int cx=-1; cx<=1; ++cx)
        {
```



Threads	Function
1	main (edge.cu:75)
32	conv2d_global (edge.cu:35)
480	conv2d_global (edge.cu:39)

DDT and ORNL

- DDT 2.4.1 installed and waiting for you!
  - "module load ddt" on Jaguarpf
  - DDT will submit job for you
- Q4 2009
  - Official DDT 2.5 release: some performance improvements
    - ~16-32k cores
  - Private ORNL access to latest DDT development
    - Much faster process control and data comparison – able to reach 100k cores and higher easily
- H1 2010
  - Further development
    - Scalable memory debugging and data export
  - DDT 3.0 stable release with performance to 250,000 cores